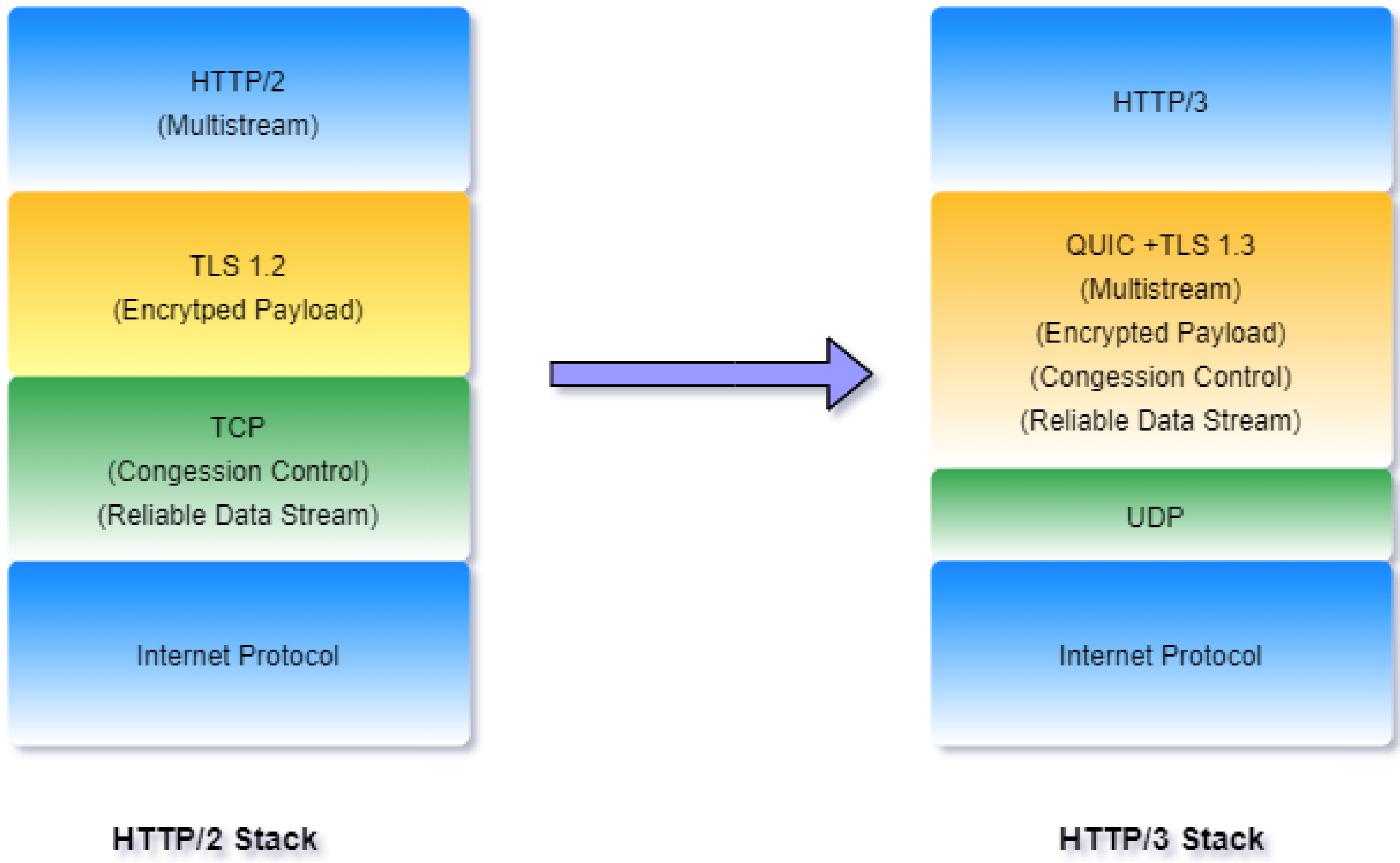
HTTP/3 with QUIC Protocols

QUIC Related RFC Standards

RFC 8999 *(was draft-ietf-quic-invariants)* **Version-Independent Properties of QUIC**	9 pages	2021-05	Proposed Standard RFC
RFC 9000 *(was draft-ietf-quic-transport)* **QUIC: A UDP-Based Multiplexed and Secure Transport**	151 pages Errata	2021-05	Proposed Standard RFC
RFC 9001 *(was draft-ietf-quic-tls)* **Using TLS to Secure QUIC**	52 pages	2021-05	Proposed Standard RFC
RFC 9002 *(was draft-ietf-quic-recovery)* **QUIC Loss Detection and Congestion Control**	42 pages	2021-05	Proposed Standard RFC
RFC 9221 *(was draft-ietf-quic-datagram)* **An Unreliable Datagram Extension to QUIC**	9 pages	2022-03	Proposed Standard RFC
RFC 9250 *(was draft-ietf-dprive-dnsoquic)* **DNS over Dedicated QUIC Connections**	27 pages	2022-05	Proposed Standard RFC
RFC 9114 *(was draft-ietf-quic-http)* **HTTP/3**	57 pages Errata	2022-06	Proposed Standard RFC
RFC 9204 *(was draft-ietf-quic-qpack)* **QPACK: Field Compression for HTTP/3**	41 pages	2022-06	Proposed Standard RFC
RFC 9287 *(was draft-ietf-quic-bit-grease)* **Greasing the QUIC Bit**	6 pages	2022-08	Proposed Standard RFC

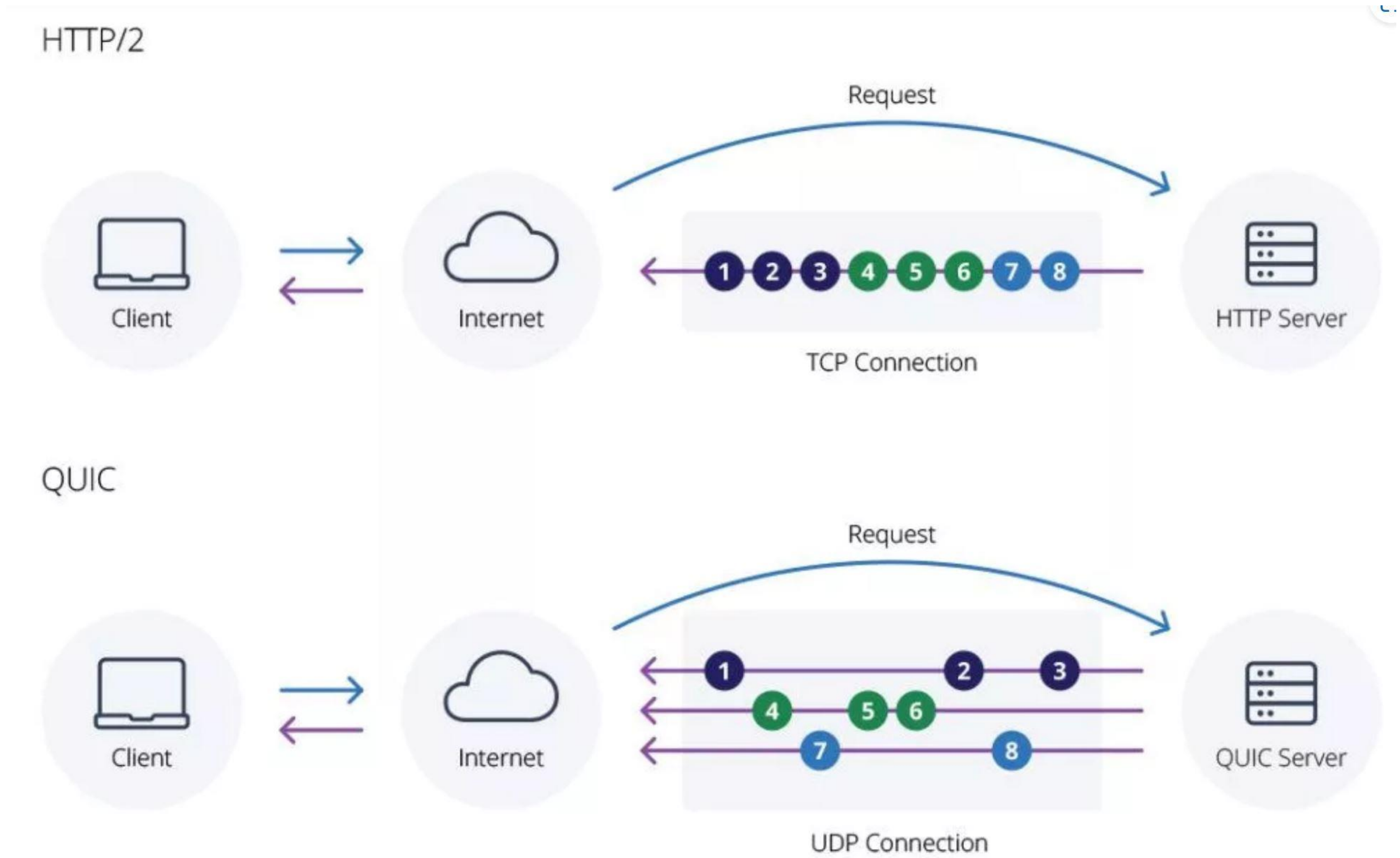
Reference: Link

HTTP/2 + TCP vs. HTTP/3 + QUIC

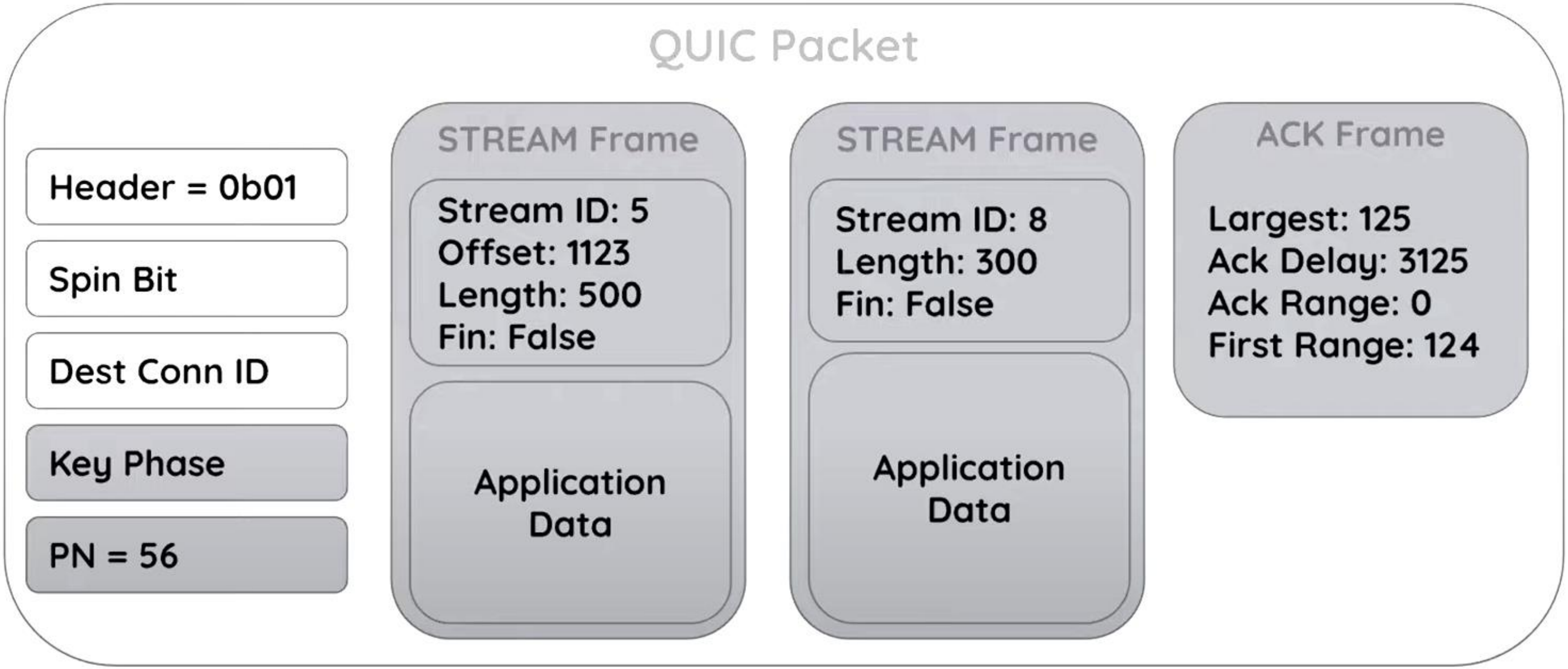


Multiple Objects Can Be Simultaneously Transferred via Streams (Interleaved)

(QUIC can solve the HOL blocking problem caused by TCP)



A QUIC Packet is Encapsulated in a UDP Datagram and Can Carry Multiple Frames of Different Types



Long and Short Header Packets Used in QUIC

1. A UDP datagrams can contain one or more QUIC packets.
2. QUIC defines two types of packet headers: long and short.
3. Long header packet (with more information and overhead) is used to establish a connection.
4. Short header packet (with less overhead) is used after a connection has been established

```
Long Header Packet {
  Header Form (1) = 1,
  Version-Specific Bits (7),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..2040),
  Source Connection ID Length (8),
  Source Connection ID (0..2040),
  Version-Specific Data (..),
}
```

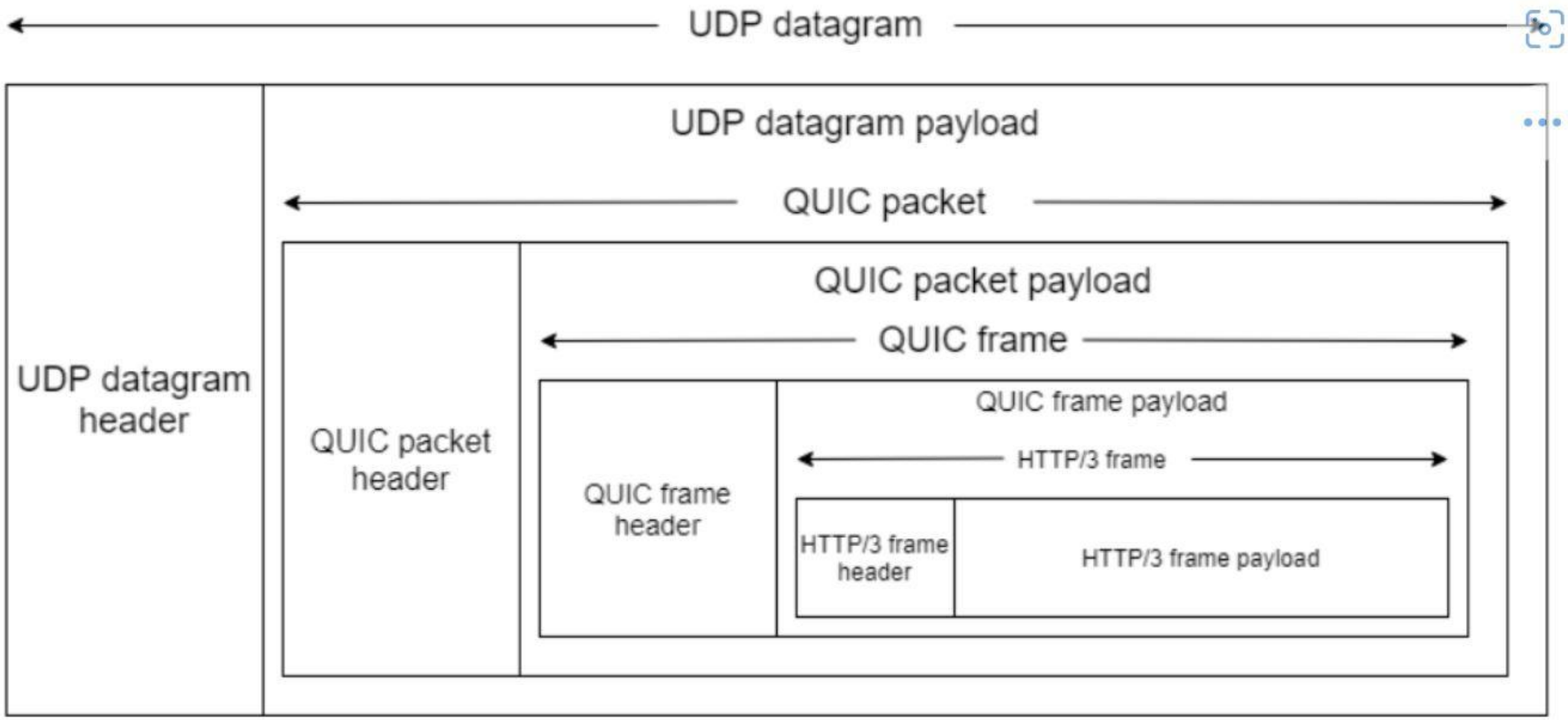
```
Short Header Packet {
  Header Form (1) = 0,
  Version-Specific Bits (7),
  Destination Connection ID (..),
  Version-Specific Data (..),
}
```

QUIC Packets and Packet Types

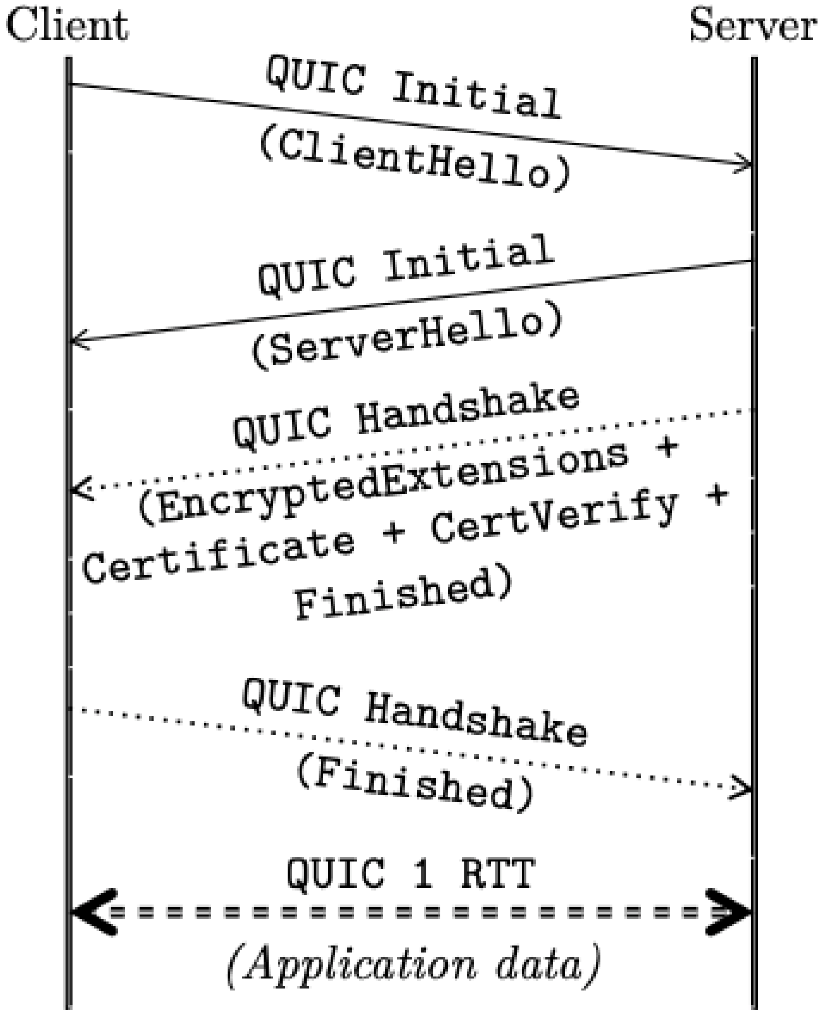
- Long header packets
  - Version negotiation packet
  - Initial packet
  - 0-RTT packet
  - Handshake packet
  - Retry packet

- Short header packets
  - 1-RTT packet

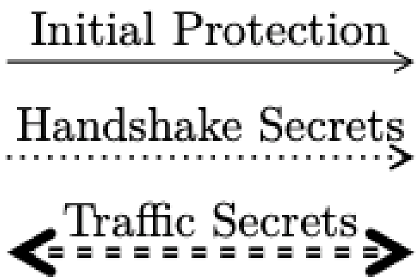
Relations among UDP Datagram, QUIC packet, QUIC frame, and HTTP/3 Frame Header, and HTTP/3 Frame Payload



QUIC Connection Establishment



1. Low-latency establishment (1RTT)
2. Stream multiplexing within a connection
3. Connection migration
4. Security



Reference: [Link]()

QUIC Connections

- The ID of a QUIC connection can be used for connection migration when the client changes its local address. For example, when the client changes its current network from a WiFi network to a 4G/LTE network.
- A connection between two peers has two different connection IDs, each of which is locally chosen by each peer.
- In a long header packet used during connection setups, there are <u>source connection ID</u> and <u>destination connection ID</u> fields.
- In a short header packet, which are used to reduce network bandwidth usage, there is only <u>destination connection ID</u> field.
- The destination connection ID is the ID locally chosen by the remote peer for a connection.
- The source connection ID is the ID locally chosen by the sender of a packet for a connection.
- Note that without using the connection ID, QUIC packets can still be delivered to the receiver by the IP/UDP address/port 5-tuple information.

Stream Types and State Machines

Bits	Stream Type
0x00	Client-Initiated, Bidirectional
0x01	Server-Initiated, Bidirectional
0x02	Client-Initiated, Unidirectional
0x03	Server-Initiated, Unidirectional

Sender

```
       o
       | Create Stream (Sending)
       | Peer Creates Bidirectional Stream
       v
   +-------+
   | Ready | Send RESET_STREAM
   |       |-----------------------.
   +-------+                       |
       |                           |
       | Send STREAM /             |
       |      STREAM_DATA_BLOCKED  |
       v                           |
   +-------+                       |
   | Send  | Send RESET_STREAM     |
   |       |---------------------->|
   +-------+                       |
       |                           |
       | Send STREAM + FIN         |
       v                           v
   +-------+                   +-------+
   | Data  | Send RESET_STREAM | Reset |
   | Sent  |------------------>| Sent  |
   +-------+                   +-------+
       |                           |
       | Recv All ACKs             | Recv ACK
       v                           v
   +-------+                   +-------+
   | Data  |                   | Reset |
   | Recvd |                   | Recvd |
   +-------+                   +-------+
```

Receiver

```
       o
       | Recv STREAM / STREAM_DATA_BLOCKED / RESET_STREAM
       | Create Bidirectional Stream (Sending)
       | Recv MAX_STREAM_DATA / STOP_SENDING (Bidirectional)
       | Create Higher-Numbered Stream
       v
   +-------+
   | Recv  | Recv RESET_STREAM
   |       |-----------------------.
   +-------+                       |
       |                           |
       | Recv STREAM + FIN         |
       v                           |
   +-------+                       |
   | Size  | Recv RESET_STREAM     |
   | Known |---------------------->|
   +-------+                       |
       |                           |
       | Recv All Data             |
       v                           v
   +-------+ Recv RESET_STREAM +-------+
   | Data  |--- (optional) --->| Reset |
   | Recvd |  Recv All Data    | Recvd |
   +-------+<-- (optional) ----+-------+
       |                           |
       | App Read All Data         | App Read Reset
       v                           v
   +-------+                   +-------+
   | Data  |                   | Reset |
   | Read  |                   | Read  |
   +-------+                   +-------+
```

Frames and Frame Types

Type Value	Frame Type Name	Definition	Pkts	Spec
0x00	PADDING	Section 19.1	IH01	NP
0x01	PING	Section 19.2	IH01	
0x02-0x03	ACK	Section 19.3	IH_1	NC
0x04	RESET_STREAM	Section 19.4	__01	
0x05	STOP_SENDING	Section 19.5	__01	
0x06	CRYPTO	Section 19.6	IH_1	
0x07	NEW_TOKEN	Section 19.7	___1	
0x08-0x0f	STREAM	Section 19.8	__01	F
0x10	MAX_DATA	Section 19.9	__01	
0x11	MAX_STREAM_DATA	Section 19.10	__01	
0x12-0x13	MAX_STREAMS	Section 19.11	__01	
0x14	DATA_BLOCKED	Section 19.12	__01	
0x15	STREAM_DATA_BLOCKED	Section 19.13	__01	
0x16-0x17	STREAMS_BLOCKED	Section 19.14	__01	
0x18	NEW_CONNECTION_ID	Section 19.15	__01	P
0x19	RETIRE_CONNECTION_ID	Section 19.16	__01	
0x1a	PATH_CHALLENGE	Section 19.17	__01	P
0x1b	PATH_RESPONSE	Section 19.18	___1	P
0x1c-0x1d	CONNECTION_CLOSE	Section 19.19	ih01	N
0x1e	HANDSHAKE_DONE	Section 19.20	___1	

- The payload of a packet that contains frames MUST contain at least one frame, and MAY contain multiple frames and multiple frame types.
- Frames of different types use different frame headers.

QUIC Error Control

- In QUIC, the data unit for loss detection and retransmission is packet rather than frame.
- Thus, each packet is associated with a number.
- Note that in TCP, each sequence number is associated with a data byte rather than a packet for loss detection and retransmission.
- The QUIC’s ACK frame is used to indicate to the sender which packets have been and have not been received yet (like SACK).
- The data frames in a lost packet will be retransmitted in a new QUIC packet with a new packet number.
- QUIC packet numbers are used to indicate the transmission order rather than the delivery order.
- The delivery order of data is determined by the offset fields of the stream frames.

ACK Frame Format

- Largest Acknowledged: representing the largest packet number the peer is acknowledging; this is usually the largest packet number that the peer has received prior to generating the ACK frame.
- First ACK range: indicating the number of contiguous packets preceding the Largest Acknowledged that are being acknowledged. That is, the smallest packet acknowledged in the range is determined by subtracting the First ACK Range value from the Largest Acknowledged field.

```
ACK Frame {
  Type (i) = 0x02..0x03,
  Largest Acknowledged (i),
  ACK Delay (i),
  ACK Range Count (i),
  First ACK Range (i),
  ACK Range (..) ...,
  [ECN Counts (..)],
}
```

ACK Range

- Each ACK Range acknowledges a contiguous range of packets by indicating the number of acknowledged packets that precede the largest packet number in that range. A value of 0 indicates that only the largest packet number is acknowledged.
- Each ACK Range consists of alternating Gap and ACK Range Length values in descending packet number order. ACK Ranges can be repeated. The number of Gap and ACK Range Length values is determined by the ACK Range Count field; one of each value is present for each value in the ACK Range Count field.

ACK Range Structure

```
ACK Range {
  Gap (i),
  ACK Range Length (i),
}
```

Figure 26: ACK Ranges

The fields that form each ACK Range are:

Gap: A variable-length integer indicating the number of contiguous unacknowledged packets preceding the packet number one lower than the smallest in the preceding ACK Range.

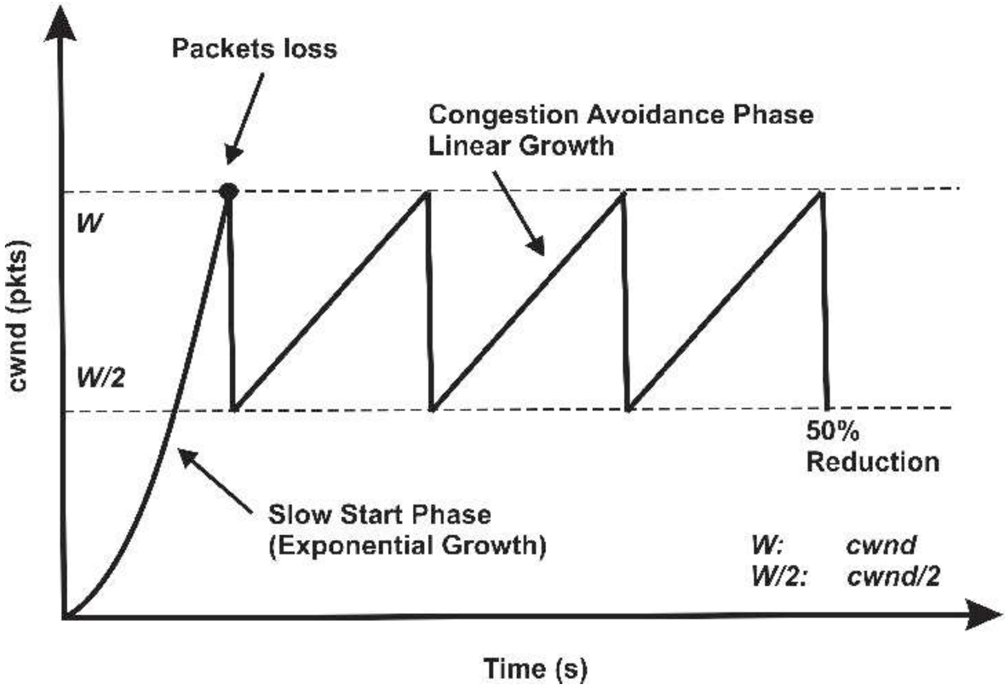
ACK Range Length: A variable-length integer indicating the number of contiguous acknowledged packets preceding the largest packet number, as determined by the preceding Gap.

QUIC Flow Control

- QUIC's flow control is applied to each stream and the whole connection simultaneously.
- The QUIC sender cannot send out a packet, which contains frames, if the connection-level or the stream-level flow control prevents it from doing so.
- Stream-level flow control is performed by the receiver sending the MAX_STREAM_DATA frame to the sender.
- The MAX_STREAM_DATA frame contains an offset in the stream data space where the sender cannot exceed.
- The sender sends the STREAM_DATA_BLOCKED frame to the receiver to indicate that it is blocked by the flow control at the carried offset.
- The information carried in the MAX_STREAM_DATA frame is not the receive window size used by the TCP receiver. Instead, it is an offset in the data space of a stream.
- The receiver will gradually advance the offset carried in the MAX_STREAM_DATA to allow the sender to send more data to it.
- Each advance in the data space offset can be viewed as a buffer space to store some incoming frame data.,
- The same flow control mechanism is applied to the whole connection by using the MAX_DATA and DATA_BLOCKED frames.

QUIC Congestion Control

- QUIC's congestion control is applied to all packets of a connection rather than to all frames of a stream.
- That is, the frames of a stream is not congestion controlled. Instead, the frames of all streams on a QUIC connection as a whole is congestion controlled.
- QUIC can use reno, new reno, or Cubic TCP congestion control algorithm to control the sending rate of a QUIC connection.




```
                 New path or      +------------+
            persistent congestion |   Slow     |
        (O)---------------------->|   Start    |
                                  +------------+
                                        |
                                Loss or |
                        ECN-CE increase |
                                        v
 +------------+     Loss or       +------------+
 | Congestion |  ECN-CE increase  |  Recovery  |
 | Avoidance  |------------------>|   Period   |
 +------------+                   +------------+
           ^                            |
           |                            |
           +----------------------------+
              Acknowledgment of packet
                sent during recovery
```

HTTP/3 QUIC Used by Different Operation Systems (June, 2022)

	HTTP/3	Non-HTTP/3
Android	47.7%	84.5%
iOS	**44.5%**	**5.5%**
Win	10.0%	5.5%
Mac OS X	1.5%	1.0%
Win7/8	1.0%	1.0%
Linux	0.3%	0.4%

Table 1 — Platforms vs HTTP/3 use.

HTTP/3 QUIC Used by Different Web Browsers (June, 2022)

	HTTP/3	Non-HTTP/3
Chrome	52.2%	91.7%
Safari	**44.6%**	**4.3%**
Firefox	2.2%	0.8%
Edge	0.6%	0.7%
Opera	0.3%	0.2%

Table 2 — Browsers vs HTTP/3 use.

HTTP/3 QUIC Packet Size Distribution (June, 2022)